

Announcements

- Change of office hours for TA Anjian Li:
 - Fridays 16:30-17:30 at ASB9810
- Assignment 2 will be posted tonight or tomorrow
- Assignment 3 will be posted tonight or tomorrow
 1. Recursion, Fibonacci sequence, and call stacks
 2. Implement sorting with recursion
 - a. Selection sort
 - b. Insertion sort

Sorting and Searching

CMPT 125

Jan. 23

Lecture 9

Today:

- Introduction to sorting
- Selection sort

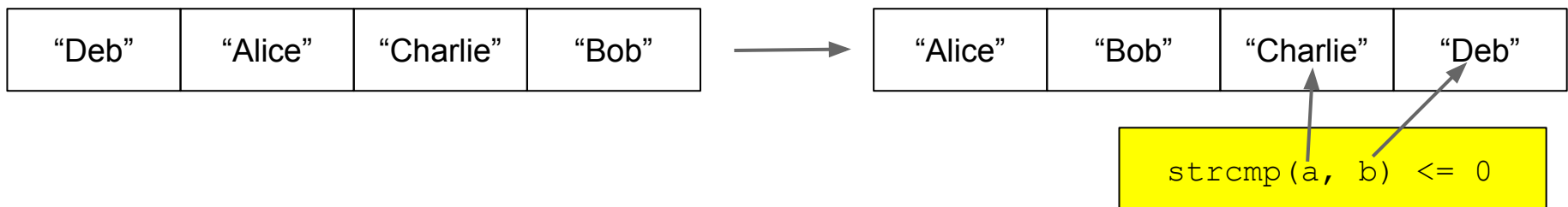
Sorting

Goal: Place a collection of items in order from smallest to largest.

- Order depends on the type of the items
- For numbers, it's by value



- For strings, it's alphabetical order



Why is Sorting Useful?

- A core algorithm in computer science, studied in depth for many many years.
 - sometimes important in its own right.
 - sometimes applied as a step in a larger algorithm.
- Our purpose isn't to learn to implement a sort
 - (but you probably will write one or two sorts in Lab)
 - many excellent implementations are out there
 - every programming language has a library sort
- Our purpose is to understand the techniques that are used and the analysis to evaluate them

Simple Sorting

- As an example of algorithm analysis let's look at two simple sorting algorithms
 - Selection Sort
 - Insertion Sort
- Predict the running time for each sorting algorithm by counting the operations
 - most expensive (i.e., frequent) operations are the comparison and movement of objects
 - compare sorting algorithms using Big-O

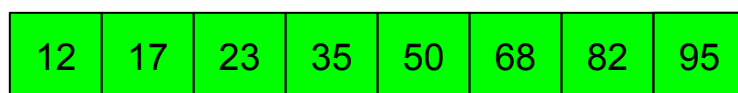
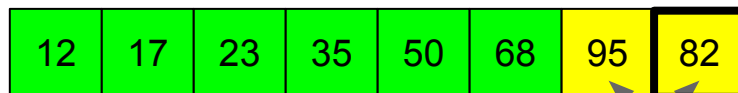
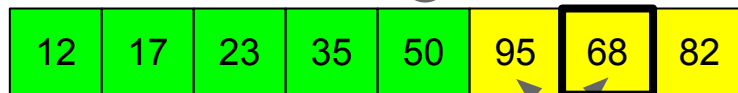
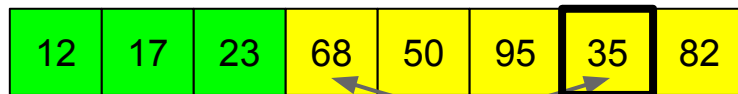
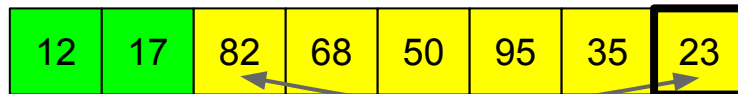
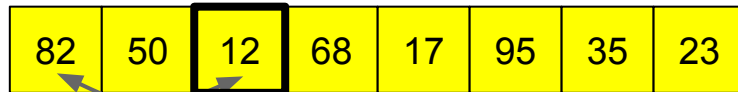
Selection Sort

Main idea: Repeatedly find the smallest item, and move it into position using a swap

- Start by finding the smallest element. Say its position is at index `minpos`
- Exchange `A[0] ↔ A[minpos]`
- Now think of the array as in two parts: a sorted part (`A[0..0]`) and an unsorted part (`A[1..n-1]`).
- Find the smallest element of the unsorted part
- Exchange it with `A[1]`
- The sorted part is now `A[0..1]`, unsorted part is now `A[2..n-1]`.
- Find the next min, . . .

Selection Sort Demo

Sort this array using Selection Sort:



find smallest unsorted item in 7 comparisons

find smallest unsorted item in 6 comparisons

find smallest unsorted item in 5 comparisons

find smallest unsorted item in 4 comparisons

find smallest unsorted item in 3 comparisons

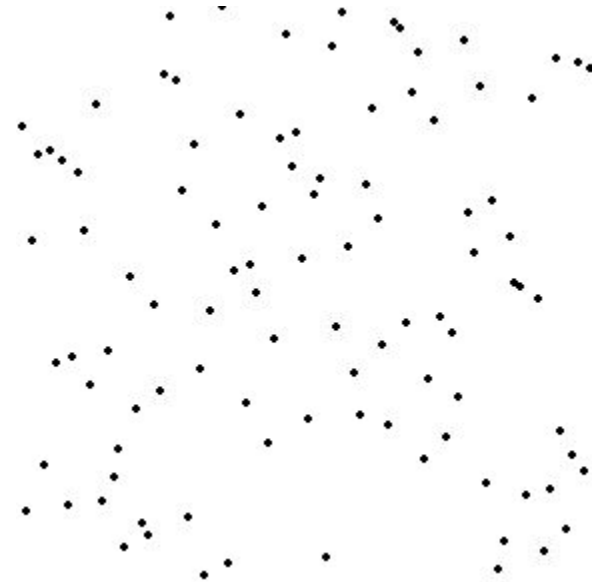
find smallest unsorted item in 2 comparisons

find smallest unsorted item in 1 comparison

find smallest unsorted item???

How many comparisons for an array of length N ?

Some Visualizations



Selection Sort in C

```
void SelectionSort(int arr[], int len) {
```

- Repeat for all `i` from 0 to `len-2`:
 - Determine `minpos`, the index of the smallest element of `arr[i..len-1]`
 - Algorithm: linear scan
 - Swap min element into position
 - `arr[minpos] ↔ arr[i]`

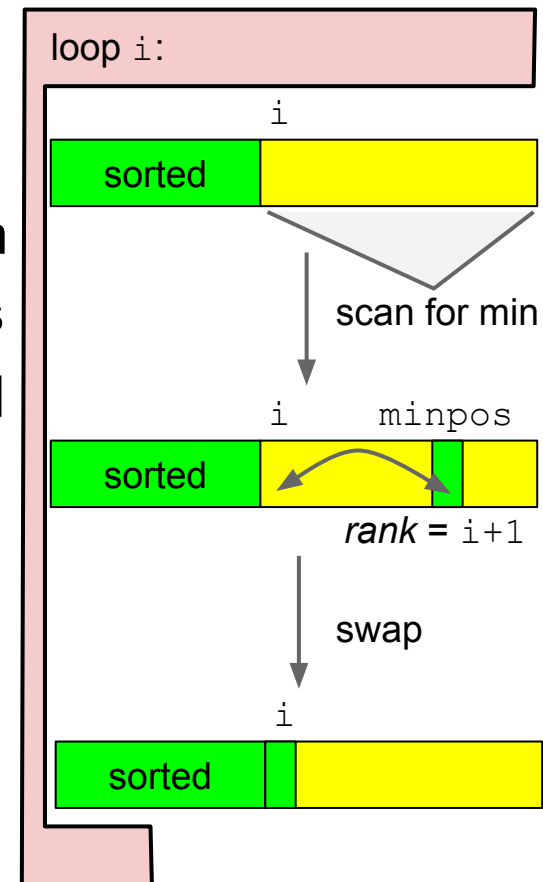
```
}
```

A Note About Assertions

Assertions: part comment / part math

Two benefits:

1. Can reason about the algorithm
 - Can visualize the progress of an algorithm
 - E.g., first i elements of `arr[]` always hold the smallest i elements of `arr[]` in sorted order
 - If you can prove the assertion holds throughout the algorithm, then you prove the algorithm is correct
 - called a *loop invariant*



A Note About Assertions

Two benefits (continued)

2. Stronger debugging

- C's `assert(condition) ;` will check your assertions, and halt your program if an assertion fails.
- a failed assertion means your program has a bug.
- parameter to `assert(...)` should produce no side-effects!

Q. What would be a good C version of the assertion used in Selection Sort?

Analysis of Selection Sort

How many steps for an input of length N ?

- Comparisons:
 - $N - 1$ to find first min,
 - then $N - 2$ for the second min,
 - then $N - 3$,
 - \dots ,
 - then 3,
 - then 2,
 - then 1.
- Swaps:
 - $N - 1$ swaps
- Total running time: $O(N^2)$

Total Comparisons

$$\begin{aligned} &= \sum_{i=1}^{N-1} i \\ &= \frac{N(N-1)}{2} \end{aligned}$$

Independent of the
nature of the input
(the initial ordering of
the numbers)

Average Case Analysis

Average-case analysis is beyond the scope of CMPT 125

- **But . . . !**
- note that many practical algorithms have an average case running time that is much much faster than its worst case.

For selection sort, the variations are small, and in all cases the running time is $O(N^2)$.

Next class: Insertion Sort!